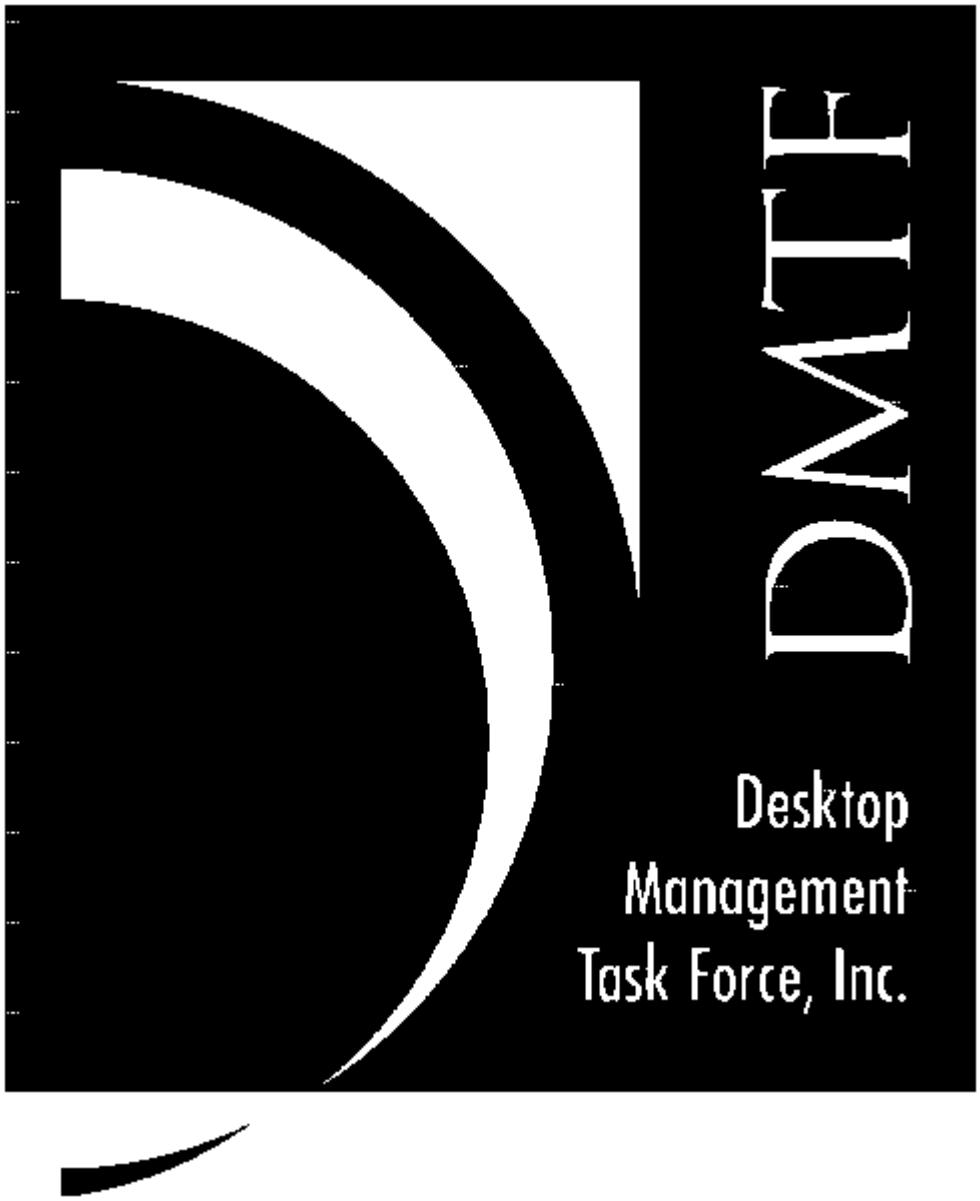
DESKTOP MANAGEMENT TASK FORCE



DMI 2.0, ERRATA #1

July 16, 1997

Draft Version 0.2 - Not Yet Approved

Technical inquiries and editorial comments should be directed in writing to:

Desktop Management Task Force (DMTF)

M/S JF2-53

2111 N.E. 25th Avenue

Hillsboro, OR 97124

PHONE: (503) 264-9300

FAX: (503) 264-9027

email: **dmtf-info@dmtf.org**

Additional electronic copies of this specification can be obtained free of charge from the Internet at:

ftp//ftp.dmtf.org

or

from the World Wide Web at:

http://www.dmtf.org

Additional hardcopies can be obtained for a fee by contacting the DMTF at the address listed above.

IMPORTANT INFORMATION AND DISCLAIMERS

CONTENTS

1. Introduction

DMI 2.0, Errata #1 describes changes and corrections made to the DMI 2.0 specification. These changes fall into the following categories:

- DMI 2.0 Document Corrections
- DMI 2.0 Interface Clarifications

DMI 2.0, Errata #1 accompanies the DMI 2.0 Interface Specification. Together they comprise the complete DMI 2.0 specification. DMI service providers and clients are expected to implement the DMI interfaces described in this errata document.

2. DMI 2.0 Document Corrections

2.1 Pragma Example Correction

Section **2.1.13** describes the Pragma MIF statement. This section correctly states that the pipe character ("|") should be escaped in Pragma strings. It is *not* legal, however, to escape the pipe character in a class string - as shown in the pragma example. Doing so results in a DMIERR_BAD_SCHEMA_DESCRIPTION_FILE error.

The following example from section **2.1.13**

```
Pragma = "Dependent_Groups: \"DMTF\|FRU\|\"; "
```

should be:

```
Pragma = "Dependent_Groups: \"DMTF|FRU|\"; "
```

2.2 MIF Grammar Correction for Unsigned Integer Types

Section **2.2** contains a BNF description of the MIF grammar. The grammar presented in section **2.2** does not allow Octal or Hexadecimal values for counter, counter64, and gauge types. This was not the intent of the specification, and revised productions for the affected parts of the grammar are presented here.

```
<Decimal Integer>   ::=     [ <sign> ] <Decimal Digit> { <Decimal Digit> } *

<Unsigned Integer>  ::=     <Decimal Digit> { <Decimal Digit> } *
| <Octal Integer>
| <Hexadecimal Integer>
```

2.3 MIF Template Correction

Section **3.4.4** provides a MIF template for the DMTF Standard Event Generation group. This template has an error in the *Key* statement. The document incorrectly states *Key=4*. The correct statement should be *Key=5*, as described in section **3.2.2**.

2.4 DmiAttributeInfo

The DmiAttributeInfo structure is described in section **5.3.3** of the DMI 2.0 Interface Specification, and appears in the **common.idl** interface description file. The IDL description is correct; the document text in section **5.3.3** is wrong and should be replaced as follows:

```
typedef struct DmiAttributeInfo {
    DmiId_t                     id;
    DmiString_t*                name;
```

```
    DmiString_t*                pragma;
    DmiString_t*                description;
    DmiStorageType_t*           storage;
    DmiAccessMode_t*            access;
    DmiDataType_t*              type;
    DmiUnsigned_t*              maxSize;
    struct DmiEnumList_t*       enumList;
} DmiAttributeInfo_t;
```

2.5 DmiFileDataInfo

The DmiFileDataInfo structure is described in section **5.3.13** of the DMI 2.0 Interface Specification, and appears in the **common.idl** interface description file. The IDL description is correct; the document text in section **5.3.13** is wrong and should be replaced as follows:

```
typedef struct DmiFileDataInfo {
    DmiFileType_t               fileType;
    DmiOctetString_t*           fileData;
} DmiFileDataInfo_t;
```

2.6 DmiSubscriptionNotice

The DmiSubscriptionNotice function is described in section **7.1.8** of the DMI 2.0 Interface Specification, and appears in the **client.idl** interface description file. The function is incorrectly described in section **7.1.8**, and is incorrectly represented in the corresponding **client.idl** file.

The sender's address was omitted from the formal parameter list, and the third formal parameter, rowData, should be typed as DmiRowData_t*, **not** DmiRowData_t. The correct function signature is as follows:

PARAMETER NAME	DIRECTION	DESCRIPTION
handle	In	An opaque ID returned to the application
sender	In	Address of the originating node
expired	In	False: Subscription expiration pending True: Subscription has expired
rowData	In	Information about this subscription. This will be the row information for the appropriate entry in the indication table defined by the "SPIndicationSubscription" group.

```
DmiErrorStatus_t  DMI_API
DmiSubscriptionNotice (
    [in]  DmiUnsigned_t         handle,
    [in]  DmiNodeAddress_t*     sender,
    [in]  DmiBoolean_t          expired,
    [in]  DmiRowData_t*         rowData );
```

2.7 DmiOriginateEvent

The DmiOriginateEvent function is described in section **8.2.3** of the DMI 2.0 Interface Specification, and appears in the **ci.idl** interface description file. The fourth formal parameter, rowData, is incorrectly typed. The correct type should be DmiMultiRowData_t*, allowing the instrumentation code to deliver multiple rows of event data.

The correct signature for the DmiOriginateEvent function is:

```
DmiErrorStatus_t DMI_API
DmiOriginateEvent (
    [in]     DmiId_t            compId,
```

```
[in]     DmiString_t*         language,
[in]     DmiTimestamp_t*      timestamp,
[in]     DmiMultiRowData_t*   rowData );
```

3. DMI 2.0 Interface Clarifications

3.1 Enumerated Attribute Values Not Within Declared Set

Section **2.1.14.4** describes the attribute value statement, including enumerated attribute types. DMI enums are { name, value } pairs that associate a text string with a specific integer value. When reading an enumerated value, however, there is no guarantee that a mapping exists for that value. Both static and dynamic (instrumented) values may be outside the range of known mappings. This means that Management Applications looking for a mapping must be prepared for the case where the mapping does not exist, and take appropriate action. For example, an application may choose to display the string representation of the ENUM value.

Note: in general it is not considered good practice to return enumerated values that are outside the known range of values, since this reduces the semantic value of the enumerated type.

3.2 Initializing Tables with a Single Row

A group definition specifying both an ID and a Key list defines an empty (zero row) table. The value statements on the attribute definitions *do not* implicitly define a table row. To initialize a table in the MIF grammar, use the MIF *table* statement, as described in section **2.1.16** of the DMI 2.0 specification.

3.3 Format Requirements for Indication Data

DMI 2.0 Service Providers perform event filtering based on the event's ComponentID, Class string, and *event severity*. The *Event Severity* value comes from the Event Generation Group, as defined in section **3.2** of the DMI specification. This means that all events must include the Event Generation Group to allow propagation of the event from the Service Provider to any management applications. If the group is omitted, the Service Provider will *not* send the event to management applications.

3.4 Format of Subscriber Addressing Field in the Indication Subscription Table

The format of the Subscriber Addressing field in the SP Indication Subscription field varies according to RPC type, Transport type, and the implementation of the Service Provider. For example, for DCE RPC and transport type ncacn_ip_tcp, the subscriber addressing information might take the form

ipaddress[port number]

where ipaddress is in dotted decimal form, and port number is the TCP/IP port assigned to the management process during its initialization.

Because the format of this field is dependent on the Service Provider implementation, itis not possible to list the formats for each combination of RPC and Transport type here. In order to remove from the burden of determining the correct contents and format of this field from the management application, Intel and IBM both provide a support function called DmiGetSubscriptionAddress(). This function may be called by a management application to obtain the subscriber addressing information for a given combination of RPC and Transport types. It takes the form:

```
DmiErrorStatus_t  DMI_API
DmiGetSubscriptionAddress (
    [in]  DmiString_t*            rpcType,
    [in]  DmiString_t*            transportType,
    [out] DmiString_t**           address );
```

Permissible values of rpcType are listed in section **3.3.1.1** of the DMI 2.0 Interface Specification. Permissible values of transportType are listed in section **3.3.1.2**.

3.5 DmiString Data Type

The DmiString data type is described in section **5.3.24** of the DMI 2.0 specification. All displayable DMI strings are null terminated, and the null terminator is included in the string length. A display string with zero displayable characters still contains the null terminator, and thus has a non-zero length. For the ISO8859-1 character format, the string length for this empty string is 1.

3.6 DmiGetMultiple Behavior

This section clarifies the behavior of the DmiGetMultiple operation as described section **6.3.3** of the DMI 2.0 specification.

When DmiGetMultiple is called without an attribute list, the Service Provider returns all attributes in the group or table row. Attributes that are UNSUPPORTED or WRITE-ONLY are omitted from the reply data, and the return status for the operation is DMIERR_NO_ERROR.

When DmiGetMultiple is called with a specific attribute list, the Service Provider returns a value for each requested attribute. Attributes that are UNSUPPORTED or WRITE-ONLY cause the Service Provider to stop processing the request and return data for all attributes up to, but not including, the error attribute.

If partial attribute data is returned, the operation's return status is DMIERR_NO_ERROR_MORE_DATA. When DmiGetMultiple returns a status of DMIERR_NO_ERROR_MORE_DATA, the caller should reissue the operation with a new attribute list. This new attribute list should start with the first attribute *not* returned in the previous call, and should contain all subsequent attributes from the original list.

If the first attribute in the attribute list is UNSUPPORTED, the Service Provider shall stop processing the request and return an error status of DMIERR_ATTRIBUTE_NOT_SUPPORTED.

If the first attribute in the attribute list is WRITE-ONLY, the Service Provider shall stop processing the request and return an error status of DMIERR_ILLEGAL_TO_GET.

3.7 Behavior When Accessing a Table without a Keylist

When accessing a table to get attributes (DmiGetAttribute, DmiGetMultiple) or set attributes (DmiSetAttribute, DmiSetMultiple), the management application may or may not specify a keylist. Sections **6.3.1** through **6.3.4** discuss these operations, but do not specify the expected behavior when the keylist is omitted. When a keylist is omitted for a table access, the Service Provider or instrumentation shall operate on the first row of the table, regardless of the Access Mode specified. This behavior is correctly described in section **8.3** of the specification.

Note that the "first row" of a table will remain constant during the execution of the SP. This is true for both instrumented and non-instrumented tables. The "first row" *can* change between reboots of the system, or restarts of the DMI SP. This restriction ensures that management applications dealing with the first row of a table are always operating on the same row.

3.8 DmiAddRow Behavior for Partially Instrumented Tables

The DmiAddRow operation is described in section **6.3.5** of the DMI 2.0 specification. This clarification extends that description to cover the case where a table contains a mix of instrumented and non-instrumented attributes. In this case, the DmiAddRow operation is not permitted, since the DMI Service Provider does not know whether to add the row in the MIF database, or in the (partially) supporting instrumentation. The DMI Service Provider will fail the operation with a DMIERR_UNABLE_TO_ADD_ROW status.

Note that, from both a design and implementation standpoint, it's generally a bad idea to mix instrumented and non-instrumented attributes in a table. This is especially true where keys are concerned. Synchronization between the component attributes and database attributes is problematic, at best. A case where some keys reside in component instrumentation and other keys reside in the MIF database is nearly impossible to implement in the Service Provider, or manage in component instrumentation. It is **STRONGLY** recommended that component providers do **NOT** mix table rows in this way.

3.9 Parsing Rules for DmiAddLanguage

The DmiAddLanguage operation is described in section **6.4.2** of the DMI 2.0 specification. This clarification describes restrictions enforced on the schema description. The description of the new language mapping must match the currently installed component's groups and attributes, excluding names, descriptions, pragmas, and values. That is the structure of the component must be maintained by the new language mapping.

3.10 Schema Definition for DmiAddGroup

The DmiAddGroup operation is described in section **6.4.3** of the DMI 2.0 specification. This clarification extends that description to define the format of the schema file. When the DmiFileType is DMI_GROUP_FILE_NAME or DMI_GROUP_FILE_DATA, the format of the data must be a valid component definition containing a single group definition. This means that the data must include both START COMPONENT and END COMPONENT declarations, and may include, for example, PATH statements and ENUM definitions at the component level. The complete MIF grammar is described in section **2.2** of the specification.

Note that certain restrictions apply to the schema supplied for DmiAddGroup():

- Table Definitions are disallowed.
- One and only one Group Definition is allowed. This group definition MUST specify a group ID (i.e., it may not be an uninstantiated template).

Schema violating these restrictions will be rejected by the Service Provider with status DMIERR_BAD_SCHEMA_DESCRIPTION_FILE.

3.11 Parsing Rules for DmiAddGroup and Multiple Languages

The DmiAddGroup operation is described in section **6.4.3** of the DMI 2.0 specification. This clarification describes requirements for the group schema definition when the component already has multiple languages installed. In this case, the fileData included with DmiAddGroup must contain a group definition for each installed language. This ensures that a *complete* language mapping is always available for a component.